Cloud Computing for Chemical Activity Prediction

Paul Watson, David Leahy, Jacek Cala, Vladimir Sykora,
Hugo Hiden, Simon Woodman, Martyn Taylor, Dominic Searson
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
{*firstname.lastname*}@ncl.ac.uk except for {s.j.woodman | h.g.hiden | d.p.searson}@ncl.ac.uk

Abstract— **This paper describes how cloud computing has been used to reduce the time taken to generate chemical activity models from years to weeks. Chemists use Quantitative Structure-Activity Relationship (QSAR) models to predict the activity of molecules. Existing Discovery Bus software builds these models automatically from datasets containing known molecular activities, using a "panel of experts" algorithm. Newly available datasets offer the prospect of generating a large number of significantly better models, but the Discovery Bus would have taken over 5 years to compute them.**

Fortunately, we show that the "panel of experts" algorithm is well-matched to clouds. In the paper we describe the design of a scalable, Windows Azure based infrastructure for the panel of experts pattern. We present the results of a run in which up to 100 Azure nodes were used to generate results from the new datasets in 3 weeks.

***Keywords**: cloud computing; chemistry; workflow*

I. Introduction

In the search for new anti-cancer therapies, the family of kinase enzymes are important biological targets since many are intimately connected to cell division and other important maintenance functions. New drugs that block the action of certain kinase enzymes are being sought by researchers at Newcastle University in collaborations with chemists, who make the new potential drugs, and biologists, who test them against the kinases. The scientists use a method known as QSAR (Quantitative Structure-Activity Relationships) [1] to mine experimental data for patterns that relate the chemical structure of a drug to its kinase activity. If a successful QSAR model can be derived from the experimental data then that model can be used to focus new chemical synthesis, and by creating QSAR models for more than one set of results, for different kinases, the new drugs can be designed to be selective.

We have an existing infrastructure - The Discovery Bus [2] - which implements an auto-QSAR (Quantitative Structure Activity Relationship) best practice modeling workflow for chemical structure property data. It can automatically generate hundreds of models for each property type and select the best and most valid. The Discovery Bus creates a library of predictive models for use in the design of better, safer drugs, as well more environmentally benign products, while at the same time reducing animal experimentation.

New databases of chemical structure properties have recently become available, covering many different types of biological action that are important in drug action, including side effects as well as environmental and human hazards. Unfortunately, building each new model is computationally intensive and so the Chemists were faced with an unacceptable five year wait to process all newly available data on their single-server Discovery Bus implementation.

However, a preliminary investigation suggested that Cloud Computing had the potential to revolutionize the use of the Discovery Bus to process new data as:

- There was the potential to generate new models in parallel, so reducing the time to process the new data.
- Compute resources were only needed irregularly, when new data was published.
- The Chemists did not have access to any other large-scale computational resources.

The "Junior" project has therefore used Cloud Computing to provide the chemists with an implementation of the Discovery Bus that allows them to quickly build new models, at an affordable cost when new data becomes available. We believe that the system is interesting for several reasons:

- Its scalable, QSAR model-building, capacity significantly exceeds that found in any other implementation: it was able to process the new data in three weeks (by exploiting 100 cloud nodes), rather than the five years the Chemists faced with their existing single-server implementation. QSAR is a common technique in chemistry and so the Cloud approach described here could have wide application. For example, QSAR is widely used for risk assessment, toxicity prediction, drug discovery and lead optimization.
- The new architecture federates resources from two clouds (Windows Azure and Amazon AWS). It also includes components written in a variety of languages. This resulted in our designing a flexible deployment system.
- The Discovery Bus has a novel, agent-based approach which is well matched to Cloud Computing; we believe that this is a generic cloud pattern could be exploited by other systems that need to generate new

models when either new data or model-building methods becomes available; we are already applying this to proving properties about software.

The paper is structured as follows. Firstly it gives a brief overview of QSAR modeling, and then describes how the Discovery Bus automates the generation of new models (Section II). It then describes and justifies the architecture of the cloud-based solution that was designed and implemented (Section III). Experimental results are then analysed for the large-scale run which consumed one hundred nodes for three weeks (Section IV). After a comparison with related work (Section V), we conclude by explaining what was learnt from the research, including how the results could be applied to other problems (Section VI).

II. The Application: QSAR and the Discovery Bus

QSAR (Quantitative Structure-Activity Relationships) quantitatively correlates chemical structure with activity such as reactivity or biological response. For example, as the number of carbons in alkanes increases, so does their boiling point: this rule can be used to predict the boiling points of higher alkanes.

If a successful QSAR model can be derived from the experimental data, then that model can be used to focus new chemical synthesis, and by creating QSAR models for different biological responses, the new drugs can be designed to be selective.

Because of the importance of QSAR, the Chemists behind our collaboration had developed a novel infrastructure - the Discovery Bus - to automatically generate new QSAR models as new data or modeling techniques became available. It could automatically generate hundreds of models for each property type, and select the best and most valid. This creates a library of predictive models used to design better, safer, more environmentally benign drugs, while at the same time reducing the need for animal experimentation.

At the highest level, the Discovery Bus can be viewed as shown in Fig. 1, where new data or model-building algorithms are inserted, triggering the generation of new quantitative models using these new data/algorithms.

Fig. 2 shows the approach taken by the Discovery Bus to generating predictive models of the activity of chemicals. The input is new data giving the activities of each of a set of chemical structures (for example the solubility in water of each chemical). The data is split into two: a training set that will be used to build models, and a test set used to validate the models (typically, 10% of the data will form the test set). A set of over 2000 molecular descriptors are then calculated for each chemical structure (an example is molecular weight). These are then merged and combined into sets of descriptors, before the CFS algorithm [3] is used to select only a subset of the descriptors: the subset is chosen to remove redundant or irrelevant features so as to speed up learning and increase the generalisability of the results. The results are then used as input by a set of model-building algorithms. Currently, the Discovery Bus implements 4 different algorithms for building mathematical models: neural networks, partial least squares, multiple linear regression and classification trees. The best models from each model-building algorithm are then compared, based on their performance on the test data. The best are then placed in the model database for use by users wishing to predict properties of chemicals. This is an example of a “panel of experts” pattern in which multiple “experts” compete to produce the best result.

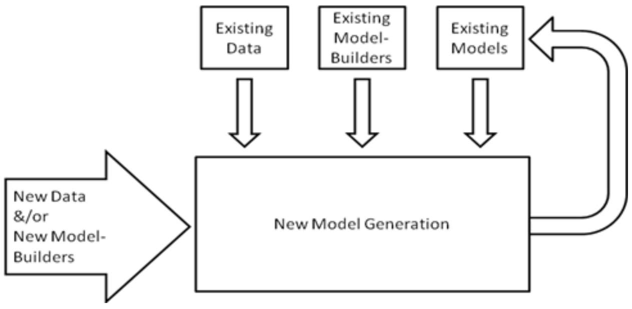

Figure 1. Discovery Bus – overall functionality

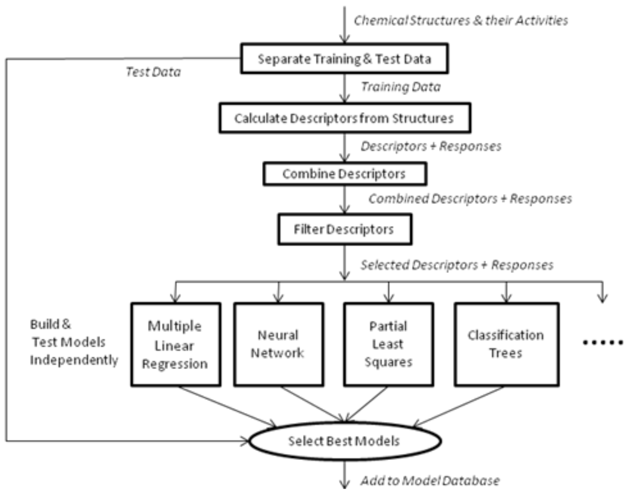
Chemical Structures & their Activities
Separate Training & Test Data
Test Data
Training Data
Calculate Descriptors from Structures
Descriptors + Responses
Combine Descriptors
Combined Descriptors + Responses
Filter Descriptors
Selected Descriptors + Responses
Build & Test Models Independently
Multiple Linear Regression
Neural Network
Partial Least Squares
Classification Trees
.....
Select Best Models
Add to Model Database

Figure 2. The Discovery Bus QSAR Process

An interesting feature of the Discovery Bus is that all the data sets are stored in the system after the model-building is complete. This means that if other model-building algorithms are introduced into the system, they can build models based on this “historic” data and, if they outperform the existing model-builders, the models they generate will be made available to users, replacing inferior, earlier models. The Discovery Bus is therefore effectively a “competitive” system in which model-building and molecular descriptors algorithms compete to produce the best models. In the next section we describe how this approach is able to exploit Cloud Computing.

III. Cloud Implementation of the Discovery Bus

The Discovery Bus is well suited to taking advantage of the opportunity offered by Clouds as:

- Processing is only needed when a new data set or model building algorithm is introduced. However when this does occur, the computational resources required can be considerable. Our interest in providing a cloud implementation was sparked when a large quantity of new data (such as ChemBL [4]) became available, which could be used to generate higher quality models, but estimates showed that this would take over five years with the existing single-server based implementation.

- There are opportunities for parallelism; in particular, each model-building algorithm can run independently. However, there are also opportunities for parallelism in the calculation of different descriptors, and the selection of sub-sets of descriptors.

A. *Initial non-Cloud Solution*

The Discovery Bus was designed with a concurrent, agent-based architecture [2] that simplified the exploitation of these opportunities on Clouds.

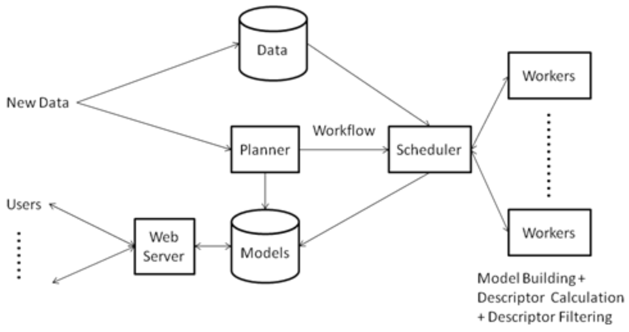

Figure 3. Discovery Bus Architecture

Fig. 3 shows the high-level architecture. Users access a website to ask questions of the models that have already been produced. These can be:

- “What are the predicted properties of compound C?” or

- “What compound is predicted to have property P?” (for example, P might be a maximum toxicity)

These questions are answered by the system extracting data from the models that it has created. The models are generated as follows. When new data arrives at the Discovery Bus, the Planner creates a workflow to control its analysis. The workflow captures the process illustrated in Fig. 2 (Section II). The workflow is passed to the scheduler which uses a set of worker agents to carry generate descriptors, filer the descriptors, and generate & test models. If the models are new (e.g. because this is the first time that a model has been generated for a particular type of activity), or outperform an existing model, then they are published into the model database for access by users.

From the start, the Discovery Bus was designed so that multiple agents could operate concurrently:

- to generate new models, using different algorithms (e.g. neural networks, linear regression …)

- to filter different combinations of descriptors

- to process datasets for different activities

However, because of the limited engineering effort available and limited access to resources, the system was running only on a single server before work to move it to the cloud began.

B. *Cloud-based Architecture*

Whilst Clouds offer the potential to revolutionize science by making large computer resources available on-demand, moving to the cloud does not make it any easier to build the complex, scalable distributed systems needed to support science. The Discovery Bus was ported over to run on the Amazon AWS cloud. When new data arrives, the Discovery Bus planner delegates the computationally expensive aspects to the Azure, for instance, descriptor calculation and model building. These parts are shown in grey in Fig. 5.

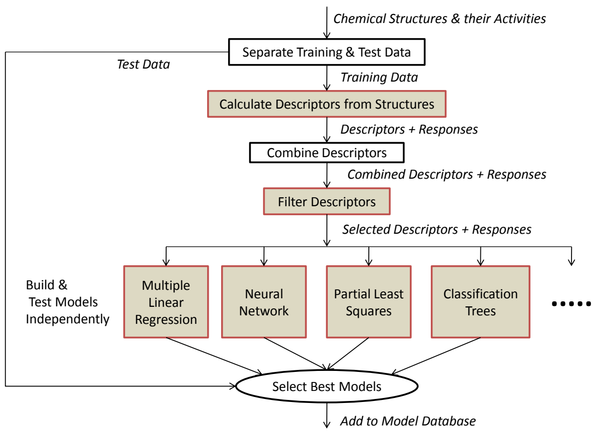
Chemical Structures & their Activities
Separate Training & Test Data
Test Data
Training Data
Calculate Descriptors from Structures
Descriptors + Responses
Combine Descriptors
Combined Descriptors + Responses
Filter Descriptors
Selected Descriptors + Responses
Build & Test Models Independently
Multiple Linear Regression
Neural Network
Partial Least Squares
Classification Trees
•••••
Select Best Models
Add to Model Database

Figure 4. The Computationally Expensive stages of model generation

We exploited the scalability of Windows Azure to accelerate the computationally intensive parts of data processing by the Discovery Bus. The Windows Azure platform is a combination of processing and storage services. The compute service is divided on two types of nodes – Web and Worker Role nodes. The Web Role is for creating applications based on ASP.NET and WCF and is the entry point for external clients to any Azure-based application. In contrast, the Worker Role runs applications as independent background processes. To communicate, web and worker roles can use the Azure storage services, which offer queue, table and blob storage. The recommended means of communication for Azure-based systems are queues. This approach facilitates scalability as many Web Role nodes can insert tasks to a queue, while others can acquire tasks from the queue. By simply increasing the number of workers, the tasks remaining in the queue can be processed faster.

This anycast processing model [5] fits well to problems that do not require communication apart from a single request-response with no state preserved in the workers. Much of the Discovery Bus processing has this kind of communication style and so it was adopted for our use case scenario.

As with many Cloud applications, we already had an existing implementation of the application – the Discovery Bus, and so we wanted to explore the best way to map this onto the Azure Cloud. There are many individual components to the application – for example all the model-generators, which are written in a variety of languages - and so rather than port each component in a time-consuming and bespoke manner we wished to find a more systematic way of deploying software onto Azure. None of the components was created in the .Net framework, but Azure makes it possible to run separate (`native') processes in an Azure-based service. We exploited this feature when designing our automatic deployment platform [7]. As components often have specific prerequisites, such as availability of Java runtime environment, the tool had to allow the expression of more sophisticated dependencies.

1) Architecture of the Automatic Deployment Solution

In order to make most of the scalability offered by the Azure cloud, our deployment platform is a queue-based system that allows a web role to request deployment of jobs on worker nodes. Every job in our approach is described by a deployment plan that defines what to install and execute. Plans are submitted by a web role *Controller* to a queue and then can be read by workers. A submitted plan is acquired by a single worker *DeployerEngine* that tries to install and execute the job. Once the execution is finished, the worker returns results to the Azure blob storage where they can be found by Controller. This communication scenario is illustrated in Fig. 6.

In the presented work *Controller* is only a façade interface for clients and uses a predefined set of deployment plan templates to describe specific Discovery Bus jobs. It provides operations that correspond to the computationally expensive stages of model generation. Conversely, *DeployerEngine* is a generic facility that allows the deployment of arbitrary, user-defined jobs. Currently, we support standalone executables, Java applications, Python, R and Windows batch scripts, but this list could be easily extended.

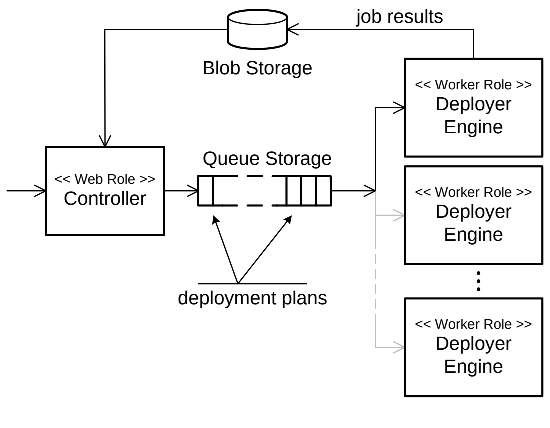

Figure 5. Communication between web role Controller and worker role Deployers.

2) Expressiveness of the Deployment Solution

One of the crucial aspects for the software deployment is expressiveness of the models used to define deployment plans. We based our solution on the D&C specification [6] that defines one of the most complete deployment standards and extended it with some additional features such as ability to express temporal constraints and support for different software technologies. As a result, a deployment plan can describe multiple tasks to be deployed, each of which may be of different type such as Java-based applications and Perl scripts. The tasks can be deployed independently or can be bound with spatial and temporal dependencies to allow the creation of sophisticated deployment scenarios. Moreover, a deployment plan may refer to other plans that need to be deployed before, which is the case for e.g. Java-based applications that require a Java runtime to be available first.

The adopted approach has proved a flexible and efficient way to move components to Azure as it avoids the need to modify existing code before deployment. To illustrate some of its most relevant features we give the following example (more details can be found in [7]).

Fig. 7 shows a simple two-task plan that executes sequentially the original Discovery Bus R script and an additional post processing program which sends results to a specific location; this plan is used in the "Filter Features" stage of Discovery Bus processing.

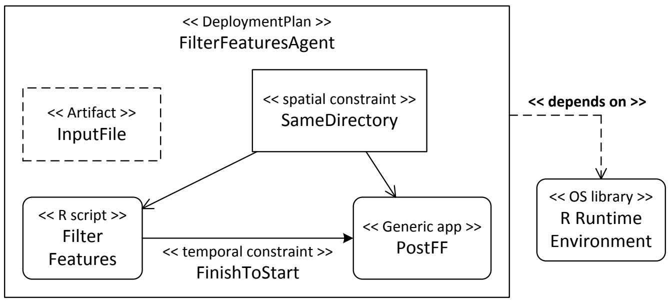

Figure 6. One of the predefined deployment plans used in processing a Discover Bus task.

The tasks are bound with the spatial constraint *SameDirectory* that requires both tasks are executed in the same working directory. The need for this constraint stems from the fact that there are implicit dependencies between them i.e. the *PostFF* task reads files produced by *FilterFeatures*. This is also the reason why we imposed the temporal constraint *FinishToStart* on these two tasks. *PostFF* cannot run until *FilterFeatures* completes otherwise some files could miss post-processing. The presented plan also shows a dependency on the R runtime environment which is not shipped with Azure and, therefore, needs to be deployed prior execution of this plan.

IV. EVALUATION

The ChemBL database [4] is a database of bioactive drug-like molecules, containing the structure of a large number of small molecules and associated biological activity. Each small molecule contains information of its activity against one or more biological targets.

In this work, the ChemBL database has been used to generate QSAR models for a number of structure-activity datasets. Initially, the ChemBL database has been curated by calculating a canonical representation of each chemical structure (canonicalised SMILES strings [8]), and producing a consistent physical unit of biological activity. Following this, in order to produce QSAR models with the capacity to be highly predictive, only datasets that contained more than 20 structure-activity values were selected, producing the number of datasets shown in Table I.

TABLE I. SIZE OF CURATED DATASETS

Database name	Number of datasets	Number of protein targets	Number of small molecules
ChemBL	8861	1138	285301

The Discovery Bus QSAR workflow has been called for each curated structure-property dataset. The Discovery Bus explores all possible combinations of workflow components, leading to an exhaustive evaluation of potential solutions [2]. Each potential solution is explored by a workflow branch. A workflow branch results when: 1) a workflow block can be computed by multiple implementations, and 2) a block produces multiple sets of results. Table II show the number of implementations and output sets given by each block in the Discovery Bus QSAR workflow.

TABLE II. NUMBER OF IMPLEMENTATIONS AND OUTPUT SETS GIVEN BY DISCOVERY BUS WORKFLOW BLOCKS

Workflow Block	Number of Implementations	Number of output sets produced
Separate Training a Test data	1	1
Calculate Descriptors from Structures	2	1
Combine Descriptors	1 (MERGE)	3
Filter Descriptors	1	6 (AVERAGE)
Build And Test Model	4	1

The Discovery Bus QSAR workflow then produces 72 workflow branches (3 *Combine Descriptors* x 6 *Filter Descriptors* x 4 *Build And Test Model*) for each dataset input given (note that *Combine Descriptors* merges 2 previous outputs from the *Calculate Descriptors* block).

Table III shows a summary of the average data size required in the QSAR workflow per dataset.

TABLE III. WORKFLOW FILE SIZES

Input Files Size	Itermediate Files Size	Output Files Size
2.8 kB	7.5 MB	1.89 MB

Using an automated process, each curated ChemBL dataset has been submitted to the Discovery Bus QSAR workflow. The workflow comprised the model building part of a QSAR, while the test (cross validation) of the generated models has been performed locally. In total, 757,563 QSAR models were generated, with 3,011 datasets having QSAR models that passed validation statistics (stable and valid). The process took a total time of 537 hrs and 18 mins to complete. Table IV shows the overall results of the computation.

TABLE IV. QSAR MODEL BUILDING RESULTS

Total computation time	537 hr 18 mins
Total number of model generated	757 563
Number of biological targets with stable and valid models	3 011
Total data transferred (in/out)	16.74 GB

In order to study the effect that moving to Azure resources has on the computation time of QSAR workflows, a set of 200 averaged-sized datasets were submitted as a batch to the Discovery Bus QSAR workflow. The number of Azure nodes was different for each batch submission (2, 5, 10, 20, 40, 60 and 80). For each batch the total time taken to compute the fully-branched workflow was recorded. Following the same configuration as with the overall computation, descriptor calculation, filter features and model building blocks were computed in Azure, each called directly from EC2. Fig. 8 shows the results of the benchmarks.

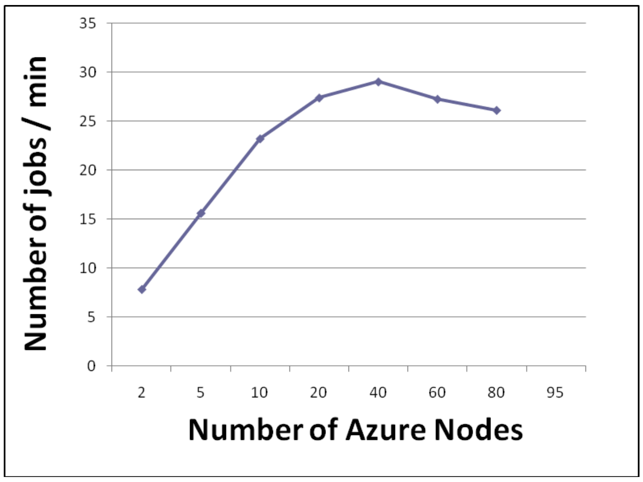

Figure 7. Job throughput vs Azure resources

Fig. 8 shows job throughput (number of jobs completed per minute) of the QSAR workflow results measured against increasing Azure resources. The figure shows the job throughput increasing with Azure resources until 40 nodes, beyond which the throughput decreases slightly with continued increases in the number of Azure nodes. We discovered that this was due to two bottlenecks: 1) the saturation of the Discovery Bus database that handles requests from working nodes, and 2) the Discovery Bus follows a centralised job planner and dispatcher, and increasing job requests from working nodes saturates the planning process.

Table V shows the exact benchmark results.

TABLE V. QSAR JOB THROUGHPUT BENCHMARK RESULTS

Number of Azure Nodes	Jobs/min
2	7.77

5	15.59
10	23.22
20	27.41
40	29.06
60	27.28
80	26.14

V. Related Work

The original Discovery Bus that was the starting point for our work is described in [2]. It was designed for a fine-grained multi-agent architecture, with the assumption that it would run on a tightly-coupled set of processors. Further, when it was written, there were not the huge amounts of data now available to build QSAR models. For these reasons it would not have been possible to produce an efficient system simply by porting the architecture to Azure. In particular, the complex and fine-grained workflow planning component was the focus of most of the effort, particularly to increase the granularity of the work it generated, and to have that work scheduled on a cloud.

A large number of existing cloud platform vendors, such as Amazon, Rackspace and RightScale, offer their solutions based on operating system virtualization. It means that to utilize cloud resources a user needs to operate on operating system images. Instead, in our approach we deploy process-level components that are much smaller when compared to an OS image. Our deployment platform allows running arbitrary jobs on any active worker irrespective of which type of job it is. This promotes better resource sharing and guarantees a more effective solution, especially for smaller and short running tasks. Moreover, our deployment plans can include many interrelated subtasks, which results in a much more expressive framework and enables the assembling of applications from existing components.

DNAnexus [9] is an end-to-end solution for sequence data management and analysis (rather than QSAR, which is the focus of this paper). It combines the scalability of the cloud with advanced sequence analysis and Web 2.0 technologies to provide an environment for DNA sequence analysis. Kepler is an environment that allows scientists to design and execute scientific workflows. Recently, Wang et al. integrated an implementation of the cloud MapReduce pattern in Kepler [10]. The "panel of experts" pattern used in this paper does not map efficiently to the MapReduce pattern, and so an alternative design and implementation is needed.

VI. Conclusions and Future Work

In Project Junior we used the Windows Azure cloud through to reduce the time taken to generate models that chemists use to predict the behaviour of molecules from 5 years to 3 weeks. In total, 750,000 new QSAR models were generated, and have now been made freely available for anyone to use (at www.openqsar.org). The nearest comparison is 50 times smaller and took nearly 20 years to collate manually. Before Project Junior, it was thought that it would be impossible to generate these models: the chemists estimated that it would take 5 years to process the vast amounts of newly-available chemical activity data that had become available. This is an ideal cloud application as large computing resources are needed, but only when new data becomes available.

Whilst the focus of the project was on accelerating the chemistry application, a series of more general lessons were learnt. For instance, the need to deploy and run existing codes written in a variety of languages on the cloud encouraged us to design a general cloud deployment system that will simplify the porting of other applications to the cloud.

In the European Union funded VENUS-C project we are further developing the system to allow greater scalability and flexibility. To increase scalability we are replacing the Discovery Bus planner which was limiting both efficiency and scalability. This is being replaced with the workflow engine from e-Science Central [11], a generic science cloud platform that we have developed at Newcastle University. We anticipate that this will bring increased performance as the engine has been designed to run in a scalable, cloud-based environment. This will allow us to extend the current work by building models from the newly available data that has been added to the ChemBL database.

The scalable, "panel of experts" algorithm implemented on Windows Azure for the Discovery Bus is an interesting pattern, well matched to cloud computing, that has the potential to be applied to other application areas. We are therefore collaborating with other scientists to apply it more widely, including for automated software checking and machine learning.

VII. Acknowledgements

The authors would like to thank Microsoft External Research for their financial and intellectual support to the project. We would particularly like to thank Christophe Poulain for the assistance he has provided, and to Savas Parastatidis for his initial efforts to champion the project.

References

[1] Hansch, C.; Leo, A. "Exploring QSAR: Fundamentals and Applications in Chemistry and Biology". American Chemical Society: Washington DC, 1995. 13.

[2] J. Cartmell, S. Enoch, D. Krstajic, D. E. Leahy, "Automated QSPR through Competitive Workflow," J. of Computer-Aided Molecular Design, vol. 19, pp. 821–833, 2005.

[3] Hall, M.A. Correlation-based Feature Selection for Machine Learning. Waikato, New Zealand, 1999.

[4] ChEMBL Team Home Page, EBI. http://www.ebi.ac.uk/chembl, July 2010.

[5] J. Abley, and K. Lindqvist, "Operation of Anycast Services," Request for Comments 4786, Best Current Practice 126, 2006.

[6] Object Management Group, Inc., "Deployment and Configuration of Component-based Distributed Applications Specification; Version 4.0," 2006.

[7] J. Cala, and P. Watson, "Automatic Deployment in the Azure Cloud," In Proceedings of Distributed Applications and Interoperable Systems 2010, DAIS 2010.

[8] V. J. Sykora, and D. E. Leahy, "Chemical Descriptors Library (CDL): A Generic, Open Source Software Library for Chemical

Informatics,” J. Chem. Inf. Mod. vol. 48 (10), pp. 1931–1942, 2008.

[9] https://dnanexus.com

[10] Wang, D. Crawl, and I. Altintas, “Kepler + Hadoop: A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems,” Proceeding sof the 4th Workshop in Support of Large-Scale Science. Portland, OR. 2009. ACM, New York, NY, 2009.

[11] Watson, P.; Hiden H.G.; Woodman S.. e-Science Central for CARMEN: Science as a Service. Concurrency and Computation: Practice and Experience, 22(17), 2369-2380. 2010.